AD-A047 477 HONEYWELL INC MINNEAPOLIS MINN SYSTEMS AND RESEARCH --ETC F/G 9/2
DISTRIBUTED DATA PROCESSING TECHNOLOGY VOLUME VI. APPLICATION O--ETC(U)
SEP 77 J BENSON, D PALMER, R STONE DASG60-76-C-0087

UNCLASSIFIED 77SRC70 NL

1 OF 1
ADA
047477

END
DATE
FILMED
1-78
DDC

DISTRIBUTED DATA PROCESSING TECHNOLOGY

DASG60-76-C-0087

FINAL REPORT

VOLUME VI

APPLICATION OF DDP TECHNOLOGY TO BMD: IMPACT ON CURRENT DP SUBSYSTEM DESIGN AND DEVELOPMENT TECHNOLOGIES

For

BMDATC
Ballistic Missile Defense
Advanced Technology Center
Huntsville, Alabama 35807

Honeywell
SYSTEMS & RESEARCH CENTER
2600 RIDGWAY PARKWAY
MINNEAPOLIS, MINNESOTA 55413

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (WHEN DATA ENTERED)

REPORT DOCUMENTATION PAGE

READ INSTRUCTIONS BEFORE COMPLETING FORM

1. REPORT NUMBER

2. GOV'T ACCESSION NUMBER

3. RECIPIENT'S CATALOG NUMBER

9

6

4. TITLE (AND SUBTITLE)
Distributed Data Processing Technology, Volume VI. Application of DDP Technology to BMD: Impact on Current DP Subsystem Design and Development Technologies.

5. TYPE OF REPORT/PERIOD COVERED
Final Report.
Oct[redacted]76[redacted] Oct[redacted]77,

14

6. PERFORMING ORG. REPORT NUMBER
77SRC70

10

7. AUTHOR(S)
J. Benson, D. Palmer, R. Stone

15

8. CONTRACT OR GRANT NUMBER(S)
DASG60-76-C-0087

9. PERFORMING ORGANIZATIONS NAME/ADDRESS
General Research Corporation
Santa Barbara, California

10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS

11. CONTROLLING OFFICE NAME/ADDRESS
Ballistic Missile Defense
Advanced Technology Center
Huntsville, Alabama 35807

11

12. REPORT DATE
Sept[redacted] 1977

13. NUMBER OF PAGES
73

14. MONITORING AGENCY NAME/ADDRESS (IF DIFFERENT FROM CONT. OFF.)

12 73p.

15. SECURITY CLASSIFICATION (OF THIS REPORT)
Unclassified

15a. DECLASSIFICATION DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT (OF THIS REPORT)
Approved for public release; distribution unlimited.

17. DISTRIBUTION STATEMENT (OF THE ABSTRACT ENTERED IN BLOCK 20, IF DIFFERENT FROM REPORT)

18. SUPPLEMENTARY NOTES

19. KEY WORDS (CONTINUE ON REVERSE SIDE IF NECESSARY AND IDENTIFY BY BLOCK NUMBER)

System, environment, and threat simulation
Software design system (SDS)
Software engineering
Performance validation
Distributed data processing (DDP)
Simulation
Verification and validation (V&V)

20. ABSTRACT (CONTINUE ON REVERSE SIDE IF NECESSARY AND IDENTIFY BY BLOCK NUMBER)

The report concentrates on one research task--assessment of the mutual impact of DDP design requirements on existing data processing design and development technologies. This research was directed into three primary areas: software engineering, performance validation, and system, environment, and threat simulation (SETS).

HD-168 REV 11/74

402 349

LB

FOREWORD

The research documented in this volume was conducted under Ballistic Missile Advance Technology Center contract number DASG60-76-C-0087, entitled "Distributed Data Processing Technology." This work was performed by General Research Corporation (GRC), Santa Barbara, California as subcontractor to Honeywell Systems and Research Center under the direction of Mr. C. R. Vick, Director, Data Processing Directorate, Ballistic Missile Defense Advanced Technology Center. Mr. J. Scalf was the BMDATC project engineer for this contract; Ms. B. C. Stewart was the Honeywell/GRC program manager.

This report covers work from October 1976 to October 1977. J. Benson, D. Palmer, and R. Stone from GRC participated in this research.

This document is Volume VI of the final report. Other volumes of the report are the following:*

Volume I	-	Management Summary
Volume II	-	DDP Rationale: The Program Planning Point of View
Volume III	-	DDP Rationale: The Technology Point of View
Volume IV	-	Application of DDP Technology to BMD: Architectures and Algorithms

*Volumes V, VI, the appendix to Volume VII, and one section of Volume VIII were prepared by General Research Corporation.

Volume V	-	Application of DDP Technology to BMD: DDP Subsystem Design Requirements
Volume VII	-	Application of DDP Technology to BMD: Experiments
Volume VIII	-	Application of DDP Technology to BMD: Research Performance Measurement
Volume IX	-	DDP Rationale: The Program Experience Point of View

CONTENTS

CONTENTS (concluded)

Section Page

LIST OF ILLUSTRATIONS

LIST OF TABLES

SECTION 1
INTRODUCTION

Distributed data processing (DDP) systems are receiving much study in both the military and general scientific communities. Originally, the studies simply focused on how to control communications so that networks of computers could "play" together. Then efforts concentrated on how to share distributed resources among a number of "users." We now recognize that the ability to distribute data-processing tasks over a network of computers offers many potential advantages--such as increased capacity (throughput), increased reliability, shorter development cycle, better growth (or change) facilities, reduced system vulnerability, architectural flexibility (to meet requirements more simply), and improved programmability and testability. Attainment of these advantages is now driving DDP design technologies, and the results are being seen in network configurations and their performance.

Another type of processing distribution is impacting many systems currently. Processing is being distributed locally (at a node) over numbers of micro and minicomputing elements. The computing architecture flexibility thus provided and the enormous advances in computing hardware technology now occurring (e. g. , the computer on a "chip" via large-scale integrated circuitry) are drastically changing computational constraints. For instance, many tasks implemented by software in the past are now implementable in hardware-providing large speed advantages and better control of reliability.

The U. S. Army Ballistic Missile Defense Advanced Technology Center (BMDATC) initiated research in FY '77 to enable obtaining the many potential DDP payoffs for BMD and to support advanced BMD system concepts. The main long-term objective of the BMDATC DDP research program is to develop and demonstrate a DDP design technology for efficient implementation and validation of BMD data-processing requirements. The main objectives of the FY '77 effort were demonstration of the feasibility of attaining long-term goals, identification of important issues to be resolved by future research, definition of longer-term research and experiment plans, and development of initial DDP design data and guidelines. Most of this research is described in companion reports.

The present report concentrates on one research task, assessment of the mutual impact of DDP design requirements on existing data processing design and development technologies. This research was directed into three primary areas: software engineering, performance validation, and SETS.* These areas are discussed somewhat separately in Sections 2, 3, and 4, respectively, although they are quite interrelated in actual designs. Some additional software engineering and performance validation aspects also are discussed in the companion report, <u>DDP Design Technology Research Requirements Definition</u>.

*SETS (system, environment, and threat simulations) programs are used as test data generators by BMDATC to test data processing design products.

SECTION 2
SOFTWARE ENGINEERING

2.1 INTRODUCTION

This section examines the effect of Distributed Data Processing systems on the software engineering process. It includes a discussion of what we consider to be the major problems in designing and developing distributed data-processing systems for ballistic missile defense, and an indication of the type of research which should be initiated in order to describe these problems in quantitative terms so that they may be solved using engineering methods. This section describes our view of software engineering by identifying the activities which comprise this process and presenting examples of what others have identified as the software engineering process. Section 3 attempts to characterize the differences between the design process for centralized data-processing systems and for distributed data-processing systems by identifying the dimensions of design freedom available in each. Section 4 identifies a number of implementation issues which must be considered when implementing a distributed data-processing system which are not usually present in a centralized system development.

Experiments to examine the issues raised have been identified. The experiments are described in a companion report, DDP Design Technology Research and Experiment Plans.

2.2 CHARACTERIZATION OF SOFTWARE ENGINEERING

Software engineering can be described as the process whereby the requirements for a system are translated into an implementation of that system in terms of computer programs. Many authors have attempts to describe a methodology or procedure for software engineering and other complex design situations. The descriptions are quite similar. For example, Freeman[1] identifies six steps in software engineering: (1) needs analysis, in which the major functions and constraints of the system are identified; (2) specification, in which the needs are converted into explicit functions and the constraints are expressed in terms of system structure and resource usage; (3) architectural design, in which the structure of the problem is identified, and from this the major pieces of the system and the relations between these pieces are described along with the basic algorithms and major data representations; (4) detailed design, in which the details of the major pieces are developed and the decision to implement the major pieces in hardware, software or firmware is made; (5) implementation, which includes the coding of algorithms and data structures, testing and reimplementation, if needed; and (6) maintenance, which includes debugging, redesign, reprogramming and enhancement of the completed system. We can summarize Freeman's and other descriptions of the software engineering process as consisting of the following types of activities, *not necessarily in the same order:*

- Functional design
- Algorithm design
- Architecture design
- Implementation
- Assignment (of algorithms to physical devices)
- Verification

Although many authors have described software engineering and similar design methodologies, very little detail has been reported--probably because of the complexity of general design problems and the difficulty of generalizing specific results. One of the most detailed and documented design methodologies is provided by the BMDATC Software Design System (SDS). The SDS methodology characterizes the state of the art of software engineering, and is described further below. Appendix A contains a more detailed outline of the SDS methodology.

BMDATC developed the SDS through research in the areas of Data Processing Subsystem Engineering (DPSE), Software Requirements Engineering (SRE), Process Design Engineering (PDE), and in specific tasks dealing with BMD software verification, validation and quality assurance. (The intent was for the DPSE, SRE, and PDE activities to be applied in sequence--with iteration--to provide the transition from system requirements to real-time software.) The research focused on providing the following key features:

1) A rigorously structured design procedure--typically consisting of a hierarchy of design/testing iterations at advancing levels of detail, the product of each level providing the specification for the next level's product.

2) Well formulated and proven design criteria for deriving products from specifications at each level.

3) The capability for constructing and exercising simulations of the product and its operational environment (e.g., SETS-type simulations) at each level.

4) A set of testing procedures, test cases, performance measures, and evaluation criteria which have known relationships (are traceable) to the specifications and design criteria, and which show the degree of success of each design level.

5) Special languages and autcmated tools to facilitate design, testing, analysis, recording, and reporting.

Our current interest concerns the relationship of SDS design procedures and criteria (i.e., the design methodology to DDP). Other portions of SDS may be relevant to DDP independently of the applicability of the methodology.

In particular, the relevancy of SDS languages, tools, and simulation and testing techniques is worthy of more detailed study. (The procedures, languages, and tools for SRE and PDE were completed to a level of somewhat general usefulness, but DPSE was not developed beyond initial studies.)

Figure 1 shows a high-level view of how the SRE and PDE phases of SDS would interact as a design methodology. The SRE and PDE phases are basically sequential, as shown, but iteration and some overlap can be assumed. Also, the system and interface specifications are considered to evolve in detail during design.

The main output of SRE, and link to PDE, is the Process Performance Requirements (PPR) specification. The SRE computerized data base for requirements development, termed the Abstract System Semantic Model (ASSM), apparently would also be available to PDE and the outside world. The ASSM is planned to contain all information for the PPR, arranged (logically) as a relational data base, and executable models of processing steps.

SRE represents functional requirements in the form of "R-nets" (requirements networks), which show processing flow (enablements of R-nets, sequences of processing steps and enabling events, and looping and branching (FOR-EACH, AND, and OR) nodes). The processing steps are termed "alphas," functional models of the steps "betas," and analytic models of the steps "gammas". Betas and gammas are used for simulations and are not intended to be part of the formal requirements specification. Software performance requirements (accuracy, port-to-port timing, and global effects)* are keyed to the R-nets through use of validation points and their descriptions. The PPR is intended to specify requirements in an implementation-free (non-constraining) context.

*Examples of global effects are: to cause a radar's duty cycle limitation to be maintained; to keep redundant tracks within limits; to keep attacker penetration of the defense (leakage) below a level; etc.

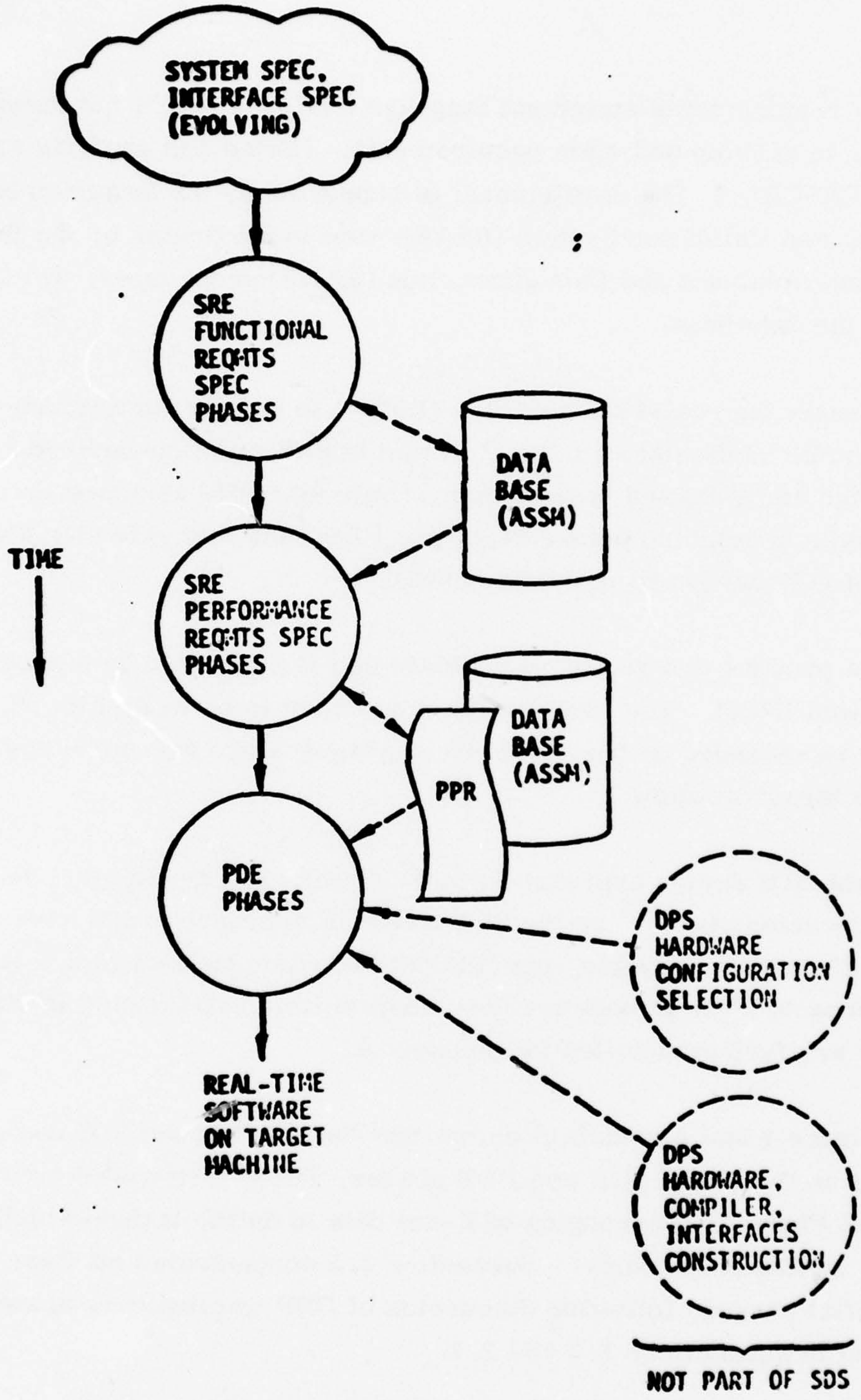

Figure 1. High-Level View of Current SDS Design Approach

SRE uses a requirements statement language (RSL), which is not directly executable, to develop and state requirements. (Betas and gammas are written in PASCAL.) The development is supported by the Requirements Engineering and Validation System (REVS)--and in particular by the REVS Requirements Analysis and Data Extraction (RADX) procedures, which enable queries of the data base.

PDE constructs the real-time software (RTSW) to satisfy application-derived requirements stated in the PPR and implementation-derived requirements, which are assumed to be known. (Note that PDE assumes that the DPS hardware is selected before beginning PDE, and that selection and construction of DPS hardware is outside of SDS.)

PDE uses a process design language (PDL) and is supported by a process design system (PDS). The final RTSW is assumed to be written in PDL, which may be extended or translated to languages which are more appropriate for a given target machine.

To define the SDS design approach in more detail, we have studied the latest available documentation[2, 3] on the SRE and PDE approaches and have identified some 15 steps (or phases, per SRE/PDE documentation) to the overall design approach. The phases are described in time relationship in Figure 2; the phases are further detailed in Appendix A.

Study of Figure 2 and Appendix A shows that the heart of the SDS design approach lies in the initial SRE and PDE phases, Phase 1 (initial definition of R-nets) and Phase 11 (regrouping of R-net data to define tasks--which are treated as virtual processors). Subsection 2.5 concentrates on these important initial phases, following discussion of DDP implications in software engineering in subsections 2.3 and 2.4.

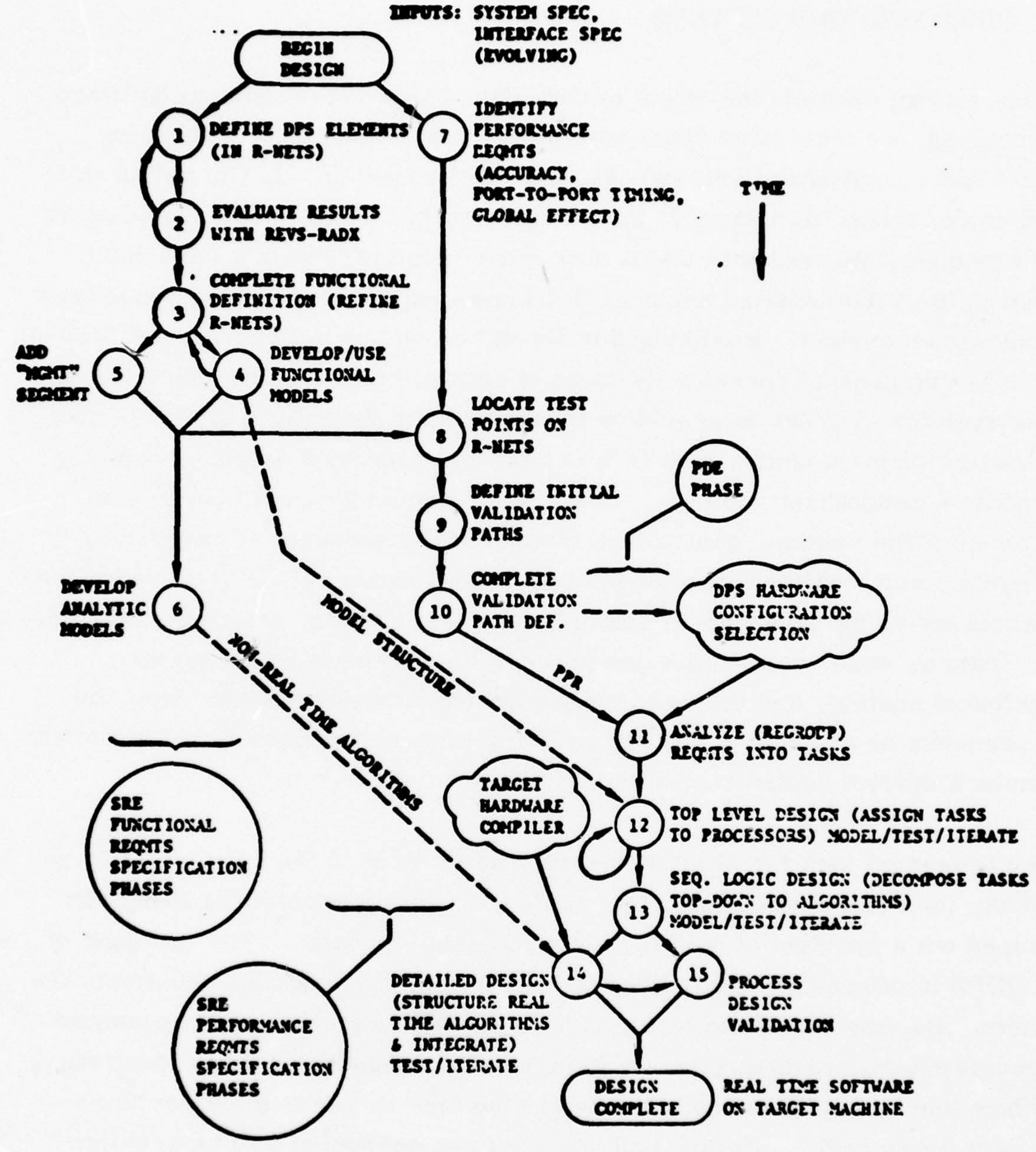
INPUTS: SYSTEM SPEC, INTERFACE SPEC (EVOLVING)
BEGIN DESIGN
1 DEFINE DPS ELEMENTS (IN R-NETS)
2 EVALUATE RESULTS WITH REVS-RADX
3 COMPLETE FUNCTIONAL DEFINITION (REFINE R-NETS)
4 DEVELOP/USE FUNCTIONAL MODELS
5 ADD "MGMT" SEGMENT
6 DEVELOP ANALYTIC MODELS
7 IDENTIFY PERFORMANCE REQMTS (ACCURACY, PORT-TO-PORT TIMING, GLOBAL EFFECT)
TIME
8 LOCATE TEST POINTS ON R-NETS
9 DEFINE INITIAL VALIDATION PATHS
10 COMPLETE VALIDATION PATH DEF.
PDE PHASE
DPS HARDWARE CONFIGURATION SELECTION
MODEL STRUCTURE
NON-REAL TIME ALGORITHMS
PPR
11 ANALYZE (REGROUP) REQMTS INTO TASKS
12 TOP LEVEL DESIGN (ASSIGN TASKS TO PROCESSORS) MODEL/TEST/ITERATE
TARGET HARDWARE COMPILER
13 SEQ. LOGIC DESIGN (DECOMPOSE TASKS TOP-DOWN TO ALGORITHMS) MODEL/TEST/ITERATE
14 DETAILED DESIGN (STRUCTURE REAL TIME ALGORITHMS & INTEGRATE) TEST/ITERATE
15 PROCESS DESIGN VALIDATION
DESIGN COMPLETE
REAL TIME SOFTWARE ON TARGET MACHINE
SRE FUNCTIONAL REQMTS SPECIFICATION PHASES
SRE PERFORMANCE REQMTS SPECIFICATION PHASES

Figure 2. Fifteen Phases of SDS Design Approach

2.3 DESIGNING DDP SYSTEMS

Before we can evaluate the effect of distributed data processing on software engineering, we must have some idea of how distributed data processing differs from centralized data processing. Unfortunately, this is not an easy thing to do, since "distributed" is a relative term. Most people would agree that a point-of-sale network was a distributed system, but is a CDC 7600? Although the 7600 contains peripheral processors, it can be characterized as a centralized system. Realizing this limitation on our definitions, we present the following general characterizations of centralized and distributed computer systems in order to provide a framework for the following discussion. In relative terms a centralized DPS is characterized by a single processing element, a centralized memory, "wide band" communication between the elements of the system, centralized control of the sequence of operations in the system and localized input and output. Conversely, a DDPS is characterized loosely by the presence of multiple processing units, distributed memories, "narrow band" communication between the elements of the system, distributed control, and dispersed input and output mechanisms. Note that the presence or absence of any one or many of these features is not sufficient to make a system centralized or distributed.

When presented with the alternatives offered by DDP to the design of a large system, the system designer is not confronted with a problem of being constrained but a problem of having too much design freedom. The designer of the DDPS is offered too many alternative ways in which he can implement the system. He may be able to locate processors and memories in any number of geographical locations, specify the type of processors at these locations and how they are interconnected. He is also free to define the algorithms and data bases of the software and how they are connected into an architecture, whether these algorithms are implemented in hardware, software or firmware, and if not in hardware, where the software that implements the algorithms will reside in the distributed system. Finally, the designer has a choice of where the locus of control will reside in a distributed system.

All these choices make up the design dimensions that the designer has to consider. On the other hand, when designing a system to run on a centralized processor, the designer does not have as much freedom in which to develop a design. The location, type and connectivity of the processor are already determined. The medium of implementation (hardware, firmware or software) may have also already been determined. Except in the case of channels or peripheral processors, the locus of control of the system and the location of the software have already been determined. Therefore, in the construction of a software system which is to run on a centralized data processor, the designer may only have to concern himself with alternative algorithms, data bases and software architectures. This is the key difference between software engineering for centralized data processing and software for distributed data processing: distributed data processing confronts the designer with many more dimensions which must be considered when developing alternative designs.

Therefore, in studying the software engineering process for distributed data processing, we must find ways in which to reduce the complexity of the design process by limiting the number of alternative designs that must be considered at any one time by the designer. There are just too many variables for the designer to consider at one time, and in addition, the alternatives in each dimension are not ordered or even organized so that rational tradeoffs can be made. Faced with this complexity, it is no wonder that tradeoffs are not apparent and ways in which alternative designs can be evaluated are not available.

The solution to this problem is to fix one or more of the alternative design dimensions and vary the others to obtain alternate designs. Note that this does not mean that we have to consider every possible design dimension. Representative examples from areas of each design dimension can be identified through the use of the nonfunctional requirements of the system. For example, if the design dimension to be varied is processor and memory placement, not all possible separations of processors and intervening geographical features need to be examined. The total set of designs in this dimension can be partitioned into those which can be assigned to categories of probability of "survivability" when subjected to a particular attack scenario. In this way representative examples of designs in a design dimension can be identified. Once these alternative designs in one design dimension can be identified, the constraints that these alternatives place on the other design dimensions can be determined. Interactions between particular alternatives in two dimensions can be identified, and the relative advantage of one design over another can be determined.

2.4 DDP IMPLEMENTATION ISSUES

The problems encountered in the implementation of a distributed data processing system can be likened to the problems which occurred when programming the first computers. In writing a program to run on one of the early computers, the programmer had to concern himself with more than the design of the algorithm and its implementation. Since higher-level languages, compilers, loaders and operating systems were not yet available, the programmer had to solve the problems of deciding where to place his program in the machine, loading his program, and starting and stopping it. Since only one program could be run at a time, he also had to arrange to schedule the machines for his own personal use. In addition, he had to explicitly store his data at machine memory locations and could only reference the data by its machine address. If input and output operations were required, he had to program these also. Many of these problems disappeared with the invention of assemblers, high-level languages and their compilers and operating systems.

Similarly, in the case of DDP, we are again faced with the problems of placement of code, access to data and startup, sequence and timing. In addition, we are no longer protected from our incomplete knowledge of the rest of the data processing system. As in the early days of programming when the programmer had to be aware of every facet and idiosyncrasy of the machine, from the instruction set to the hardware that implemented each instruction, the designer of a program to run on a distributed system must be aware of the architecture of that system. Previously, higher-level languages like FORTRAN permitted the programmer to ignore the mass of detail concerning how the computer was constructed and the other programs which were running on the machine. He was given an abstraction, a FORTRAN "virtual machine," in which to describe the solution to his problem, the program. Unfortunately, distributed data processing systems have no such high-level virtual machine architecture which the designer of the applications programs can use. Therefore, during the software engineering of a program for a DDPS, the problems of placement or assignment of program and data to machines, access to data, sequencing and timing of processes and knowledge of the rest of the system must be explicitly taken into account.

The designer must (in the absence of an assignment or loading function) decide where each piece of software will reside in the distributed system. This raises many questions for the software engineering process. First, how should programs and data be assigned to processors? One program to one processor? One program to many processors? Or many programs to one processor? In some cases, for survivability reasons, the designer may decide that programs should migrate from one processor to another in the distributed system. Another question concerns the rationale for assigning programs to processors. Should the algorithm be tailored to fit the processor or the processor designed to fit the algorithm? What evaluation criteria should be used? Should execution speed be the criterion for assigning programs to processors and access time the criterion for assigning data bases to memories or are there other important reasons for making particular assignments in a distributed data processing system? All these decisions were not present or were greatly simplified in the software engineering of a centralized data processing system.

The location of data bases and whether they are centralized or distributed is another question that the software engineering process must address. Data access will also be a problem in a distributed system. A mapping function must be designed for each data item which identifies not only in which memory location the item resides, but also where the memory resides. For the case of distributed data bases, the problems of simultaneous update and the maintenance of duplicate copies of information must also be addressed during the software engineering process.

Problems in sequencing and timing involve the synchronization and coordination of parallel processes, the cooperation and mutual exclusion of these processes when they are accessing shared data, the prevention of deadlock and race conditions and the demonstration of the correctness and repeatability of the asynchronous processes in a DDPS.

Knowledge of the other components of the DDPS is especially critical in the design and development of distributed data bases and distributed control functions. Allocation and scheduling decisions may have to be made using incomplete knowledge of resources, processes, and other decisions made by controllers in the distributed system. The critical questions involve the implementation and verification of the correctness of such a system. These questions and those raised in the previous paragraphs must be answered during the software engineering process for a DDPS.

2.5 TECHNOLOGY ASSESSMENT

2.5.1 Overview

Previous sections develop the point that conventional software engineering does not provide an adequate methodology for treating DDP design requirements, but it does contribute representational techniques and tools of rather general usefulness. More specific analyses are necessary for a more quantitative technology assessment. Consequently, following subsections specifically consider the applicability of the software design system (see *Subsection* 2. 2 and *Appendix A--to* DDP).

2. 5. 2 Discussion of SRE Applicability

The SRE design approach is shown in Figure 1 (Subsection 2. 2) as sequencing two basic activities: define functional requirements and define performance requirements for BMD data processing (software). The first activity, definition of functional requirements, produces R-nets. The second activity keys performance requirements to the R-nets.

The major questions of applicability of SRE phases to DDP design can be thus stated as follows:

1) *Applicability of Functional Requirements Definition Activity:*

- Does any reason exist why R-nets cannot be constructed to fully define DDP functional requirements, assuming that they can always be constructed for conventional DP designs?
- Would R-nets constructed for DDP functional requirements in any way mislead or undesirably constrain lower level design phases?

2) Applicability of Performance Requirements Definition Activity:

- Does DDP design contain any performance and/or design requirement considerations that cannot be keyed to R-nets? (If so, is another method of statement possible and satisfactory?)

These questions are discussed below, following more detailed descriptions of certain fundamentally important SRE phases where necessary.

The fundamental step in R-net definition occurs in SDS Phase 1. Figure 3 details Phase 1, as interpreted from Reference 2. We see from Figure 3 that R-nets are first established for each input interface to the DPS from external subsystems. The initial structures are determined by the interface message contents hierarchies. Then, details of the structure are derived from the system specification through a combination of tracing functional threads, constructing data hierarchies for external entities with which the DPS must deal, "sentential" decomposition (associating alphas with verbs and data with nouns in the source spec), defining the need for independent data files (files not attached to the interface message or entity data hierarchies), and restructuring to show potential asynchronous processing.

The Phase 1 activities all require a degree of creativity and presuppose the ability to develop source specification details (with the customer's concurrence) as necessary to proceed. This type of activity is likely to be the same for distributed as for nondistributed data processing design. The sources of interface and DP functionality definitions, which are the sources of R-nets, should be no different.

The question of R-nets being misleading or constraining depends a good deal on the ability of lower-level process and architecture designers to restructure R-net data to suit their concepts. If R-net restructuring requires excessive "creativity" at lower design levels, then the R-net construction portion of the SRE approach would not be considered applicable for DDP design.

A part of SRE which potentially could mislead/constrain lower level design is Phase 1.4.8, restructuring to show asynchronous processing; this phase would seem to require more creativity where distributed architectures are being considered.

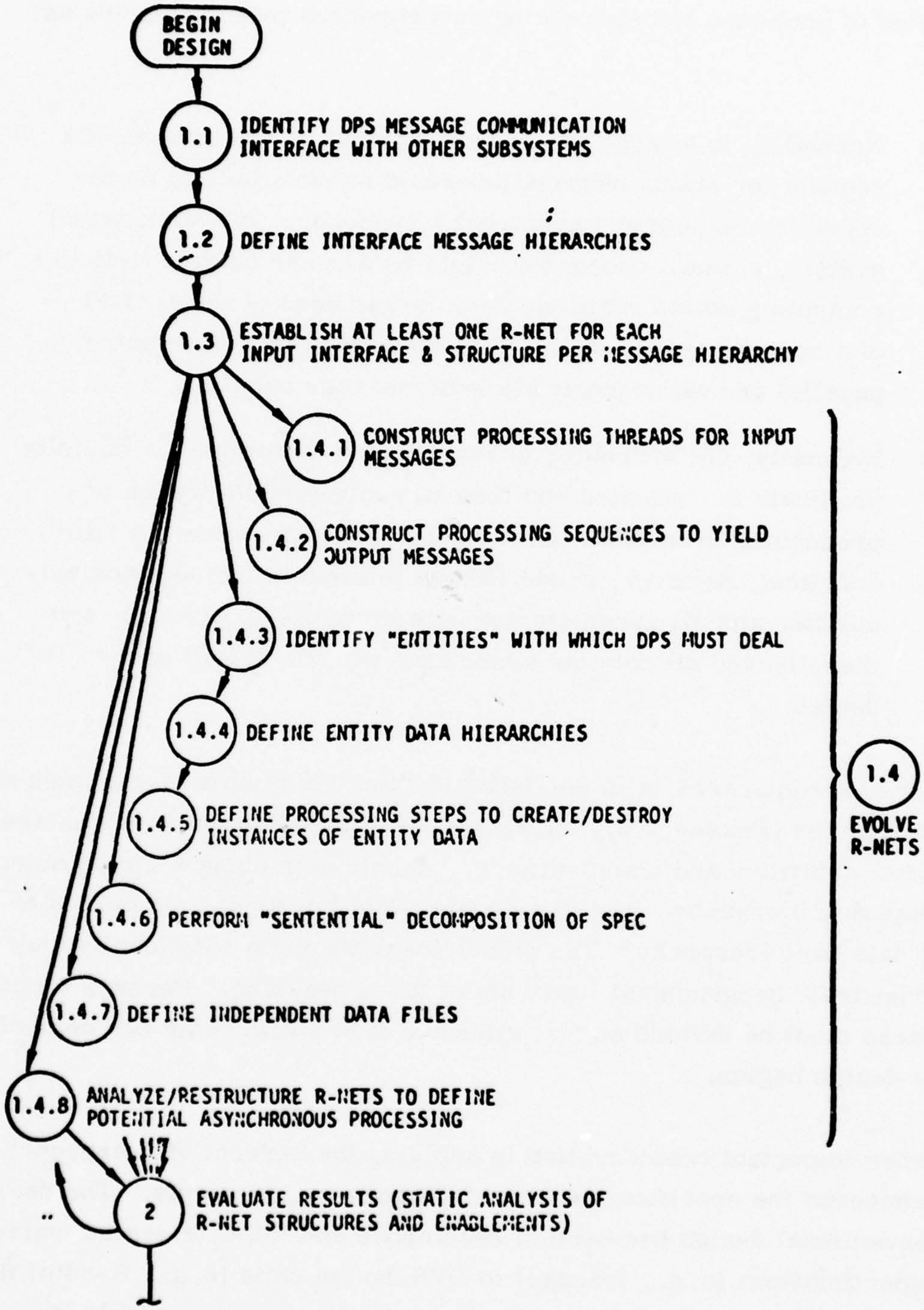
BEGIN DESIGN
1.1
IDENTIFY DPS MESSAGE COMMUNICATION INTERFACE WITH OTHER SUBSYSTEMS
1.2
DEFINE INTERFACE MESSAGE HIERARCHIES
1.3
ESTABLISH AT LEAST ONE R-NET FOR EACH INPUT INTERFACE & STRUCTURE PER MESSAGE HIERARCHY
1.4.1
CONSTRUCT PROCESSING THREADS FOR INPUT MESSAGES
1.4.2
CONSTRUCT PROCESSING SEQUENCES TO YIELD OUTPUT MESSAGES
1.4.3
IDENTIFY "ENTITIES" WITH WHICH DPS MUST DEAL
1.4.4
DEFINE ENTITY DATA HIERARCHIES
1.4.5
DEFINE PROCESSING STEPS TO CREATE/DESTROY INSTANCES OF ENTITY DATA
1.4.6
PERFORM "SENTENTIAL" DECOMPOSITION OF SPEC
1.4.7
DEFINE INDEPENDENT DATA FILES
1.4.8
ANALYZE/RESTRUCTURE R-NETS TO DEFINE POTENTIAL ASYNCHRONOUS PROCESSING
1.4
EVOLVE R-NETS
2
EVALUATE RESULTS (STATIC ANALYSIS OF R-NET STRUCTURES AND ENABLEMENTS)

Figure 3. Details of SDS Phase 1 Activities

Examples of problems in determining asynchronous processing are as follows:

1) Normally, in a large centralized computer, the arrival of a request for status information would cause a task to be invoked to compute and construct a response. In a distributed system, a small processor might be devoted continuously to computing status response data, regardless of the arrival of a requesting message. This task could therefore occur in parallel and asynchronously with message handling.

2) Normally, OR branching is represented as follows: a decision predicate is evaluated and then an exclusive OR branch of processing is invoked based on the predicate value. A DDP architect, however, might choose to execute the decision calculation and all candidate outcome branches in parallel, and then discard all data not associated with the result of the decision.

Another hazardous area is in the definition and use of data hierarchies and their attributes (Phases 1.3, 1.4.4, 1.4.7). Some data hierarchies are somewhat arbitrary and transient; e.g., should data about a threat object be a separate hierarchy, part of a booster data hierarchy, or part of a threat data base hierarchy? The global/local attribute additionally may later constrain geographical locations of the processing. Perhaps virtual interfaces must be defined so that interface data hierarchies can be partitioned before design begins.

The other important consideration in applying the current SRE approach to DDP concerns the specification of performance requirements. The desire for conventional design has been to decompose and allocate system performance specifications (e.g., leakage) to DPS design units (e.g., R-nets) thereby providing firm performance criteria for each unit to be used in lower-level design. The ability to obtain this result has not been generally realized, and now the development of less directly applicable criteria is being proposed; SDS Phases 7-10 describe the approach.

The current SDS approach consists of identifying performance requirements, locating test points on R-nets where the performance can be checked, and defining sets of validation points at which data is to be collected to support the test. (The types of performance requirements being considered are herein termed to be accuracy, port-to-port timing, and "global effects.") This approach will usually lead to the definition of performance requirements which are spread over a series or combination of R-nets, often involving many stimulus/response sequences to define. Such a specification cannot be expected to substantially aid lower-level design, but it does provide test criteria for the nearly completed process design.

The specification of SRE-type performance requirements for DDP must be at least as difficult as for conventional systems. In addition, DDP design is intended to consider performance and design requirements which are quite beyond the current SRE-types; e.g., graceful growth, fail soft, reconfigurability, reliability/availability, vulnerability, etc.

The inclusion of some DDP requirements might be accomplished by defining new functions (e.g., reconfiguration fault detection/correction) and establishing associated R-nets. Other requirements might be accommodated through incorporation of design evaluation models in R-nets (e.g., a graceful growth evaluation model might be a function of the number of interfaces, amount of interface data, enabling events, and global data linkages of specified R-nets); during development, the R-net configuration should be evaluated with the same type of model. Other requirements, such as vulnerability, will require architectural constraints on the dynamic relocatability of specified functions to be specified. (The current RSL language apparently can support extensions of the types described.)

2.5.3 Discussion of PDE Applicability

The PDE approach is predicated on the DPS hardware configuration being preselected. It is intended to map the PPR functional requirements onto the hardware in an efficient manner, which hopefully will satisfy the PPR performance requirements regarding accuracy, port-to-port timing, and global effect.

The primary question of PDE applicability to DDP design is whether the DDP hardware configuration can be preselected or whether an integrated hardware/software design is necessary to attain requirements efficiently. An integrated design would be expected to contain a different set of considerations and tradeoffs than treated by the current PDE approach.

Secondly, assuming that the DDP hardware could be selected or characterized in detail before process design, would the PDE approach be sufficiently flexible for process design for a wide range of DDP architectures and according to DDP performance and design requirements? If so, for what design aspects would the DDP approach be applicable?

We will proceed for the case that the hardware configuration can be preselected or characterized in detail. Then, the applicability of PDE is determined mainly by SDS Phase 11 (PDE Phase 1), which is, in part (per Appendix A):

> "Establish a traceable mapping of the PPR Requirements Nets into data processing tasks structured to effectively utilize the architectural characteristics of the real-time data processor.
>
> In particular, assuming a defined hardware configuration, restructure the R-nets to group processing steps (alphas) into tasks on the basis of commonality of:

- Suitable processor types and capabilities (where the hardware configuration is assumed to contain more than one type of processor)
- Scheduling frequency and conditions
- Memory organization (assuming a memory capability hierarchy to exist)
- Potential parallelism of different steps acting on one data entity instance and potential concurrency of identical steps acting on many instances of a data entity (assuming the hardware configuration to contain more than one processor architecture)
- Real-time operating system requirements: tasking and data hierarchy, scheduling fidelity, and memory management.

This activity may produce more than one candidate process configuration."

Later, in SDS Phase 13 (PDE Phase 3) a top-down decomposition of the tasks is conducted as follows (see Appendix A):

"Viewing each task as an isolated program in the system to be placed into execution on a particular processor, decompose each task into subtasks on the basis of commonality of:

- Input/output
- Timing and scheduling
- Logic characteristics

Prescribe the sequencing (or decision) logic for selective execution of the subtasks of each task. Test and iterate to lower levels."

The approach described above appears to be broadly applicable but does not specifically consider many DDP requirements. The requirements, per the DDP SOW are indicated in Table 1; also see the companion report, DDP Design Theory Research Requirements Definition.

TABLE 1. DDP DESIGN REQUIREMENTS AND CONSIDERATIONS

1. Node (Process) Characterization for Purposes of Partitioning BMD Application Operations for Assignment to Nodes (Processes). Considerations Include:
   - Node vulnerability
   - Control
   - Time response
   - Data base
   - Communications
   - Throughput
   - Reliability
   - Rules for distributing control
     - Consistent with BMD application control structure
     - Static and dynamic assignment of BMD operations to nodes (processes)
   - Internode (interprocess) communications
     - Protocol
     - Node (process) integrity (e. g. , fault detection and correction)
     - Node (process) redistribution or fault tolerance
     - Node (process) sanity

2. Networks to Connect Nodes (Processes)
   - Structure/resources at nodes (processes)
   - Link structure (e. g. , path definition and management)
   - Network integrity (detect/correct/reconfigure)

3. Distributed Data Base Design Considerations (Partitioning/Assigning Data)
   - Initial and dynamic relocation
   - Data integrity (accuracy, time validity, redundancy, security, error detection/correction)

4. Performance Issues During Design, Test, Validation, Maintenance, and Engagement
   - BMD operations
   - Node (process) capacity
   - Performance response time
   - Communications
   - Node vulnerability
   - Reliability/availability
   - Validation and sets concepts

Comparing PDE phases and DDP requirements, we see the current PDE approach does not explicitly treat most DDP requirements: e.g., communications, reliability/availability, data integrity, and networks. Furthermore, the PDE approach does not specify in sufficient detail how to characterize processes and hardware configurations for grouping of R-net requirements, and how to perform control and data base design.

2.5.4 Summary of SDS Applicability Assessment

The results of studying the applicability of SRE phases to DDP are as follows:

1) The SRE approach to defining functional requirements and their representation as R-nets is not basically incompatible with high-level DDP requirements specification concepts. More study is required to determine whether details of the approach might mislead or undesirably constrain lower-level design. These details involve the recognition and representation of potentially asynchronous processing and the creation and use of hierarchical data structures.

2) The SRE approach to defining performance requirements, although possibly the best available in the DP community, appears to provide little firm design guidance for lower level design (even for conventional DPS design). In addition, the approach does not treat most of the types of requirements of concern in DDP design; e.g., vulnerability, fault detection/correction etc.

3) The SRE language and tools enables constructing a requirements data base for designing a wide range of systems. These components may be quite valuable for DDP design, independent of the applicability of the SRE requirements specifications approaches.

The preliminary results of studying the applicability of PDE phases to DDP are as follows:

1) PDE does not integrate software and hardware designs, which are considered necessary for effective DDP design.

2) Even for the case that hardware selection need not be integrated with process design, the PDE approach offers almost no help to DDP design except at very low levels. The PDE approach does not explicitly treat many DDP requirements (e.g., communications, reliability/availability, etc.) has provided insufficient detail in treating most other important requirements (e.g., processing task definition, control, data base design).

3) The PDE language (PDL) and tools facilitate expressing top-down software designs, and should be applicable for any process design where PDL can be translated into the target machine language.

SECTION 3
PERFORMANCE VALIDATION

3.1 INTRODUCTION

"Performance validation" is sometimes interpreted in a very broad sense; e.g., it may include proving logical correctness with respect to axioms and rules, validating logical correctness with respect to assertions in given environments, verifying correctness of logic during tests, showing achievement of designed performance (e.g., efficiency) during tests, and some other combinations of these ideas. The object to be "validated" also is often ambiguous; e.g., it may be the final design product implemented (SW/FW/MW combination), some part of the final product implementation, a simulation of the final product at some level of abstraction, a system containing the final product implementation, etc.

Our main concern here is being able to "validate" that design activities at each level have produced "correct" results. Consequently, we herein construe "performance validation" in the following way:

1) During design and development of a BMD system, many "design products" are generated. These design products formally represent the system at various abstract levels; i.e., they represent the system concept, requirements, configuration (architecture), etc., as design progresses.

2) The DDP design activities contribute "DDP design products" to the BMD system design products.

3) Performance validation then is construed to be the demonstration that DDP design products at each level (a) represent or satisfy all DDP requirements, and (b) enable associated BMD system design products to represent or satisfy all system requirements.

4) All requirements are assumed included: functionality, control, data, communications, performance (timing, accuracy, capacity, etc.), and "payoffs" (survivability, reliability, etc.)

5) The means of demonstration includes all applicable techniques: showing consistency, static and dynamic structural tests, environmental testing, etc.

The extensiveness of performance validation indicates the need for much organization of activities for efficiency and completeness. We must characterize design products and requirements and define appropriate validation (demonstration) techniques at each level. Requirements for supporting tools, test drivers, and procedures can then be developed.

The characterization of products and requirements is discussed in subsection 3.2. In subsection 3.3 the mutual impact of DDP and existing performance validation technologies are assessed. Much of the impact is shown to be associated with the design and use of SETS, which is treated in Section 4.

3.2 CHARACTERIZATION OF PRODUCTS AND REQUIREMENTS

During software engineering for a centralized or a distributed data processing system, a number of different products must be evaluated and tested. Specifications and requirements may be in the form of natural or formal languages. In either case these specifications must be evaluated for consistency (both among themselves and with previous requirements), completeness (in terms of previous specifications), and correctness. Computer programs in the form of functional, analytic, and hybrid simulations or real-time software must also be evaluated. Algorithms, whether simulated or actual, must be evaluated in terms of their correctness, accuracy, and performance. In the development of distributed data processing systems, computer hardware or firmware may also need to be evaluated. In the evaluation of hardware or firmware components, the detection and isolation of faults and the measurement of performance are principal goals.

We further characterize design products, requirements, and validation by describing techniques used in the BMDATC SDS (see subsection 2.2). The techniques are organized into the following activities: Hierarchical Verification and Validation (HV&V), Adaptive Verification and Validation (AV&V), and Advanced Software Quality Assurance (ASQA).

The objective of HV&V was to assure the efficient integration of comprehensive software reliability and testing requirements into the DPSE, SRE, and PDE programs (see subsection 2.2), and to develop the overall testing and management methodologies for use of the testing tools and procedures developed within the programs. HV&V had responsibility for: (1) special testing for early error correction and final real-time software (RTSW) acceptance; (2) construction and use of special test support software (H-SETS and H-PEP)*; and (3) control of information flow, and coordination of testing, within the software development process itself and between the data processing subsystem and other subsystems. The software development system was to provide a computer-processable representation of the developing software at each level, both at the major (DPSPR, PPR, RTSW) levels and at the incremental levels between. HV&V identified four kinds of testing to which these products were to be subjected: system performance testing, code performance testing, performance allocation testing, and development integrity testing.

*H-SETS = Hierarchical System, Environment, and Threat Simulation.
H-PEP = Hierarchical Performance Evaluation Processes.

The AV&V program was concerned with the automation of the testing of BMD software. Its major goal was to identify specific categories of threats which would cause the BMD system to fail to meet its performance requirements. In the case of AV&V, threats were characterized by a set of threat parameters such as the number of RVs and decoys, and the collection of the values of these threat parameters defined a threat space. Threats (as described by specific values of the threat parameters) which cause the BMD system to fail to meet its performance requirements define a "limiting performance contour" in the threat space. The goal of the AV&V program therefore was to identify this performance contour automatically. To do this, AV&V has constructed computer programs which are collectively referred to as the "adaptive tester." A functional diagram of the adaptive tester is shown in Figure 4. Using the Interactive Scenario Construction function, the analyst specifies the initial test inputs in the form of a scenario. Using a model of the BMD system, an engagement is automatically simulated. (Models of both a terminal defense and mid-course defense are available.) Measures of the performance of the system (the objective function) are automatically calculated and recorded. Finally, the parameter perturbation algorithm automatically alters the scenario in an attempt to maximize or minimize the objective function, and reruns the engagement. Current parameter perturbation algorithms are gradient, random, complex, heuristic and grid search.

The goal of the ASQA program was the development of a methodology for the formal verification of large real-time systems written in contemporary programming languages. This included the development of tools for detecting errors, assisting testing, and performing formal verification and the evaluation of existing and proposed programming languages. The methodology for program verification developed by the ASQA program is shown in Figure 5. It includes steps for locating and correcting syntax, semantic and execution errors in real-time systems and a technique for showing formal correctness of programs. These techniques have been implemented in a collection of computer programs called the Software Quality Laboratory. The tools

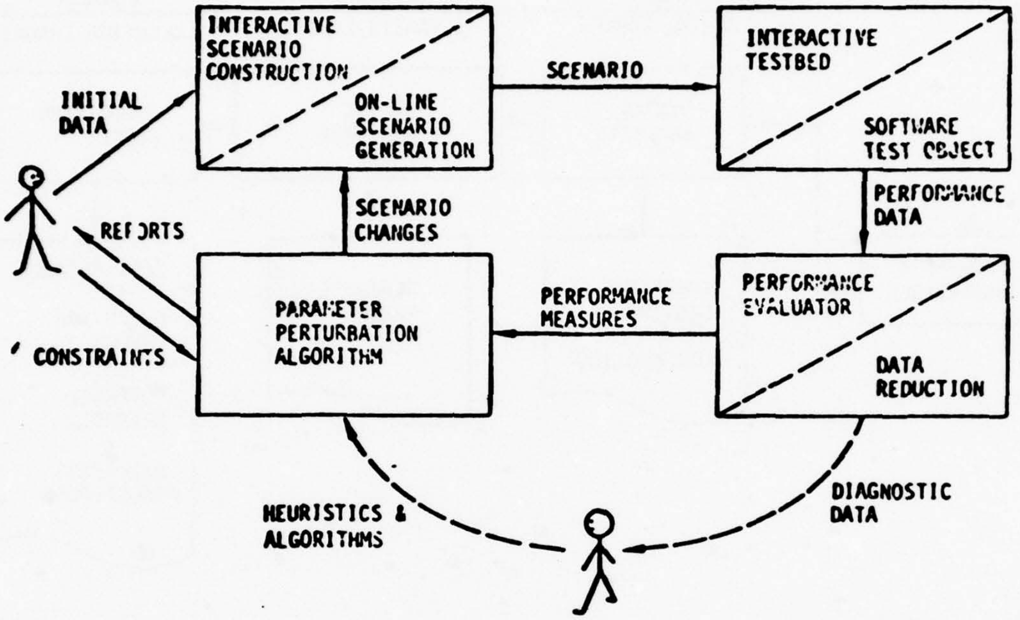

Figure 4. Adaptive Testing Functions

in the Software Quality Laboratory can be applied to two programming languages: IFTRAN and Verifiable Pascal. These languages are extensions to existing languages (FORTRAN and Pascal) which were developed by the ASQA program in order to improve the reliability of programs written in these languages. Preprocessors were developed for both languages which automatically generate executable code from assertions stated in programs written in these languages. These executable assertions can be used to detect errors at execution time. Static analysis tools, which use the assertions and other redundant information in the IFTRAN and Verifiable Pascal languages, to identify errors were also developed. Formal program verification is implemented by a verification condition generator which automatically uses the assertions in a technique called symbolic execution to develop

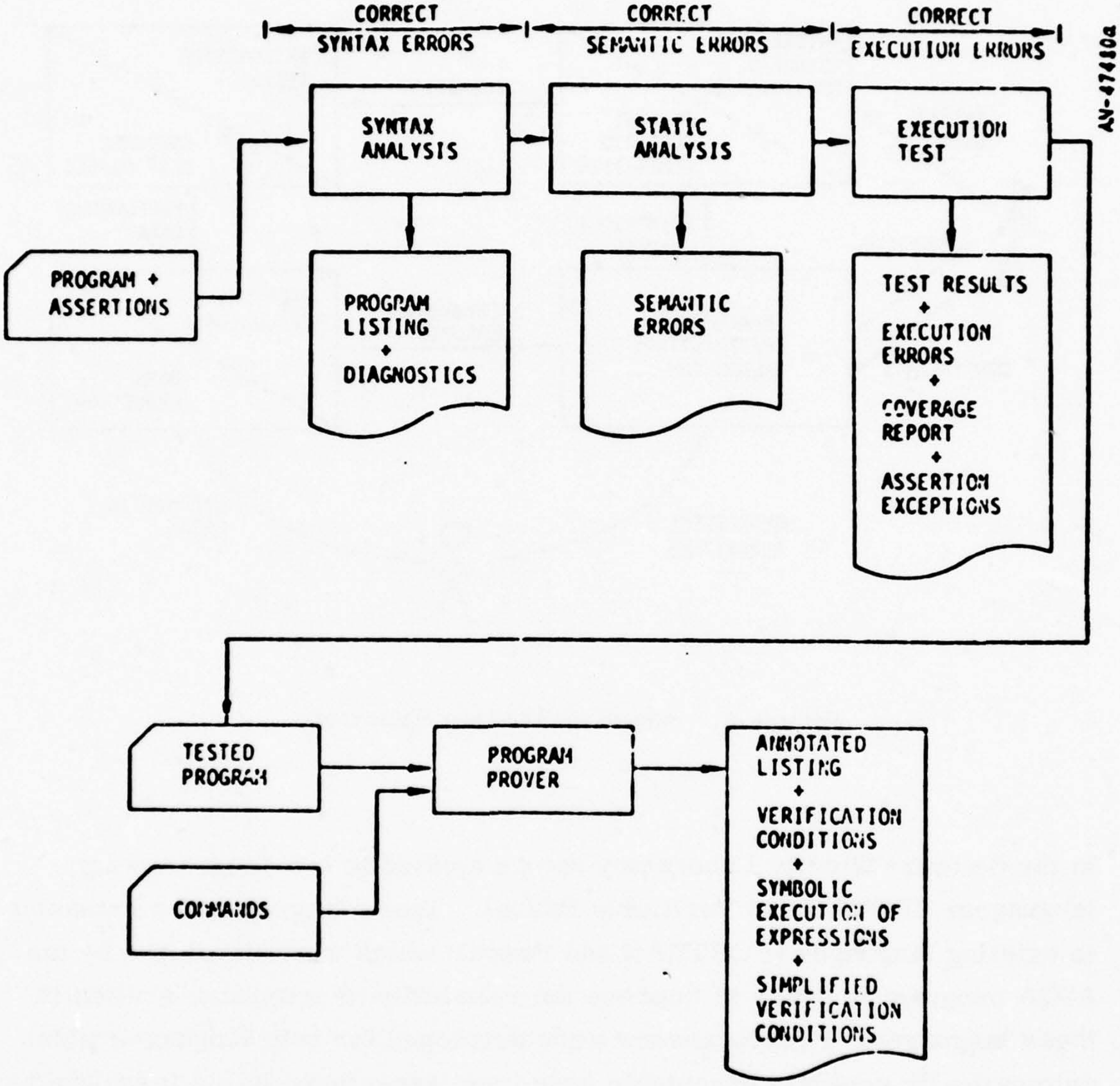
CORRECT SYNTAX ERRORS
CORRECT SEMANTIC ERRORS
CORRECT EXECUTION ERRORS
AN-47480a
SYNTAX ANALYSIS
STATIC ANALYSIS
EXECUTION TEST
PROGRAM + ASSERTIONS
PROGRAM LISTING + DIAGNOSTICS
SEMANTIC ERRORS
TEST RESULTS + EXECUTION ERRORS + COVERAGE REPORT + ASSERTION EXCEPTIONS
TESTED PROGRAM
PROGRAM PROVER
COMMANDS
ANNOTATED LISTING + VERIFICATION CONDITIONS + SYMBOLIC EXECUTION OF EXPRESSIONS + SIMPLIFIED VERIFICATION CONDITIONS

Figure 5. Steps in Verification Procedure

formal statements concerning the correctness of a program. These formal statements are then simplified with the aid of an automated simplification program to prove the correctness of the program.

3.3 VALIDATING DDP SYSTEMS

In some ways, the evaluation of distributed data processing systems is quite similar to the evaluation of centralized data processing systems. The requirements for performance evaluation and testing, for example, are nearly the same distributed systems; however, by their nature they greatly complicate the process of performance evaluation and testing.

Evaluating the performance of a distributed system involves three activities: the identification of performance measures, the development of methods for evaluating the performance measures, and relating the performance measures to data processing and system requirements. Performance measures can be identified at many levels of the system. At the highest level they are directly related to system performance (e.g., the number of penetrating RVs), whereas at the lower levels they are only indirectly related to the overall performance of the system (e.g., bus timing). Evaluating these performance measures therefore requires data collection mechanisms at all levels of detail in the system. High-level performance measures require taking measurements of the system as a whole, whereas low-level performance measures require measurement techniques which are more local in nature. In addition, the low-level performance measures usually have a greater probability of disturbing the action of the system. It is also difficult to determine which measurements should be taken and how these measurements are related to overall system performance. Performance measures must be identified which relate to all types of requirements--functional requirements such as system capacity, correctness and accuracy, requirements for correct sequence of operations and timing and nonfunctional (unstructured) requirements such as reliability and survivability. A key problem is developing low-level performance measures whose relation to high-level unstructured requirements can be determined.

3.4 DDP VALIDATION ISSUES

The very nature of distributed systems causes problems in testing them. The physical distribution of processors and sensors requires that measurement-gathering mechanisms be distributed also. This implies increased cost and time in completing the testing effort. In addition, there are many more things to measure in a distributed system as opposed to a centralized system. Multiple processors, communication links and distributed data bases all require data to be gathered concerning their performance. One cost advantage of distributed architectures, the use of simple processors, may tend to increase the difficulties of testing and data collection. There may not be enough available processing time or memory in the small processor to support data collection and fault-detection algorithms unless this is provided for during the design of the system. Since a distributed system is implemented using various types of components, hardware, firmware and software, it may be difficult to differentiate which type of component is responsible for errors. In addition, different testing and data-collection mechanisms are required for each type of component.

The development of a central data collection function will also be difficult in a distributed system. Since large amounts of performance data will have to be collected, ways of preprocessing, compressing and communicating this data to a central data collection function will have to be developed. This implies extra burden upon the processors and data links of the distributed system which should be considered at system design time. In order to identify the sequence of events in the distributed system a central clock will have to be implemented. All significant measurements will have to be related to the central timing mechanism since we would expect the individual parts of the distributed system to operate asynchronously.

3.5 TECHNOLOGY ASSESSMENT

Research issues in performance measurement and testing of distributed systems include the design of a distributed system, environment, and threat simulation (distributed SETS), the identification of the problems associated with a centralized performance data collection function and the development of a testing method for distributed systems. The design of a distributed SETS is discussed in Section 4. The problems involved in a centralized performance data collection mechanism are mentioned above. The development of a distributed testing methodology is discussed further here.

The major problem in testing a distributed system is the system's complexity. Distributed systems are composed of many dispersed components, much more closely linked with their local environment than with the rest of the data-processing system. The time, cost and effort required to test such a system will be far greater than that required to test a centralized data processing system. For this reason, the testing and performance evaluation procedures for distributed systems must be highly structured and well coordinated. In fact it very well may not be possible to test the whole system together, especially if data processing capability is to exist in remote sites which are only part of the distributed data processing system for discrete time intervals (e.g., satellites and interceptors). For this reason, individual components of the system would be tested independently, with other parts of the system being simulated or emulated. Models of the entire system would then be tested using techniques similar to those developed by AV&V. For this type of testing to succeed, methods for relating the results from testing individual components to the complete system must be developed. In addition, a coherent order in which to test individual system components must be designed. Finally, ways of developing and of verifying the accuracy of system models need to be investigated since the entire system may never be subjected to testing at one time.

SECTION 4
SETS

4.1 INTRODUCTION

In previous sections design products are identified as they would appear in a distributed system development cycle. Some type of SETS-like test data generator is required for products at most levels. The mutual impact of DDP and the SETS technology mainly occur only when distribution actually is implemented, however. Following discussions therefore address SETS issues at very low design levels almost exclusively.

Until recently, predominating interests in the defense community have been oriented toward defenses in which large centralized computers carried out most of the real-time control logic. The resulting technology studies emphasized techniques for designing, building and testing software to operate in this environment. With the current interest in distributed global and local defenses controlled by networks of coordinated computers previous work must be re-examined to accommodate the resulting new dimensions. Among the software requiring extensive redesign is the System, Environment and Threat Simulation (SETS) program used as the environment simulator in testing.

The SETS program concept has been used successfully by BMDATC and other agencies to test various forms of BMD software items. As indicated above, test configurations have generally simulated terminal defense constructs in which the real-time software resided in a single large machine. For simplicity, and because hardware existed for that purpose, SETS too was designed to operate in a single large computer.

With the distribution of the real-time process over many computers, the task of providing timely environmental information simultaneously to many processors becomes extremely complex. The major issues relating to SETS testing involve coordination of timing among the various real-time processes and determining the appropriate SETS program configuration to operate in a distributed environment. It is the purpose of this section to assess the impact of distributed systems on SETS. Experiments that, if accomplished, would help resolve critical issues are suggested in a companion report, DDP Design Technology Research and Experiment Plans.

4.2 ISSUES AND TECHNOLOGY ASSESSMENT

A number of very difficult problems confront the designer of a SETS system for use in a distributed data processing development effort. We have discussed how past SETS programs have operated in a dedicated large-scale, general-purpose computer in a serial mode interfacing with real-time processes in a similar but separate machine. When we start to examine distributed systems, problems which were difficult but surmountable in previous configuration appear to be nearly impossible to mold into solvable form. Foremost among these problems is the manner in which time is synchronized between SETS and the real-time processes (RTP).

Where in the past, we were concerned with a single process running at any instant in the SETS program and in the real-time code, now multiple processes will be operating simultaneously in the RTP and possibly within SETS. The associated synchronization problem within the RTP, within SETS and between the two has not been studied extensively and certainly no general solution has yet been presented.

In Subsection 4.2.1 some background is presented on the interrupted real-time mode used in past testing efforts along with a discussion of the implementation issues associated with real-time, interrupted real-time and repeatable real-time operation in a distributed system.

The question of how to configure the SETS program itself in the presence of distributed real-time components is the next to be addressed. Some issues arising from a centralized SETS design responding to multiple-data input streams are discussed (Subsection 4.2.2). While it might appear that the only possible way to test a distributed real-time process is with a distributed SETS system, this approach is fraught with disturbing pitfalls, some of which are presented.

Finally, some general topics are discussed relating to the role of SETS in the testing process and how best this role might be fulfilled. Alternate testing modes are examined and a foundation is laid for a set of experiments suggested in later sections.

One strategy proposed to guarantee real-time operation requires a resource supervisor to monitor the SETS process and to switch to low-fidelity models in the face of impending overload. The advantage of this mode of test is that real-time could be maintained at all costs. However, the degradation in fidelity could result in unrepeatable test runs, a distinct disadvantage during a software development cycle.

Closed-loop operation is required for those situations when SETS cannot predict the sequence of commands to be given to it. For a completely general program it is highly desirable to build SETS in this fashion so that a change in the command logic of the RTP* will not require a corresponding change in SETS. The cost of this generality, however, is quite high. SETS must be able to respond to arbitrary orders in an arbitrary sequence. (Of course an order must be part of a previously well-defined repertoire of orders.) Preprocessing of data can be carried out to facilitate preparation of returns; however, to compute complete returns ahead of command time

*Real-time process.

would require a massive quantity of storage, knowledge of the command sequence, as well as a completely predictable real-time process.

Preprocessing in past SETS configurations has concerned itself with building tables used for indirect addressing into processed threat data. Sky tables were generated which rapidly converted a commanded beam direction into a sequence of pointers to data for objects in the general area of the beam location. The object data was then retrieved, its current location precisely determined (by evaluating a polynomial especially fit over a segment of the trajectory) and a computation carried out to determine the object's contribution, if any, to a sensor return. Response time can be enhanced by preprocessing but only to a certain limit. One must be cautioned against preprocessing to the degree that closed-loop testing becomes impossible: that is, when SETS no longer can easily respond to changes in RTP logic. Moreover, the resources required to input, sort, process and present the preprocessed data may well exceed that required to perform the calculations on-line.

4.2.1 Time Control in Testing with SETS

Multiple processes working with a common base of time-varying data or which are otherwise interchanging data must have a means of remaining synchronized in time. In postulated distributed BMD processes, highly reliable and accurate time sources have been assumed to reside within each system component. The problems of maintaining coordination of synchronization between the processes have not been completely resolved. The presumption is that the time sources are sufficiently accurate that drift-free operation can be assumed. Some multi-static systems require that the receiving radar know the time of a pulse transmission to within a few tens of nanoseconds. How this information is made available to the receiver has not yet been spelled out in detail.

If we place a distributed system in a test environment using SETS as the environment simulator the problem of process time coordination becomes important since it is highly unlikely that the SETS process can maintain real-time operation. The problem arises because the SETS calculations usually cannot be performed with sufficient fidelity in the short amount of time allocated to it. That is, to supply environmental information, radar returns, object information, any data required by any subset of the real-time process is a detailed and time-consuming process. In the past the SETS process has relied upon many devices to enhance its response time, notably extensive scenario preprocessing to enable it to quickly assemble data needed for a specific calculation. However, no amount of preprocessing or even computing power can guarantee that SETS will always be able to prepare its responses within time constraints. This has been true in terminal defense single-site simulations and it will certainly be true with multiple distributed processes. What has permitted testing in the past to be carried out in a realistic manner was the concept of "Interrupted Real Time" (discussed on the following pages after an initial look at alternative approaches to achieving real-time operations).

SETS in Real Time -- As we stated above, SETS is generally unable to carry out its calculations in real time, for two reasons:

1) "High fidelity" models are used
2) Closed-loop operation is required.

The term "fidelity" can take on many meanings in our context. We might represent the threat quite grossly in terms of arrival rate, radar cross section, aggregation of many objects into a single representative but could couple this with a detailed sensor model which carefully computes range mark data. This configuration is an unlikely one and when we use the term "high fidelity" or "low fidelity" we assume generally a broad consistent level of modeling over the spectrum of SETS programs.

The fidelity with which SETS carries out its calculations and presents its results must be consistent with the information content to be derived from those results. This is a basic axiom of testing using an environmental simulation. It generally turns out to be easier to model phenomena at high fidelity than at low fidelity. For example, the equations governing the formation of radar returns from the propagation and reception of radar signals are well known and readily computed. While lower-fidelity models may run faster and supply statistically correct responses, there is a limit to their utility whenever real-time algorithms are to be tested for correctness. Little research has been done to support development of such models. So, while it is conceivable that real-time capability of SETS can be enhanced through the use of low-fidelity, fast-running models they must be developed with the proviso that at least as much information content must appear in their responses as is to be derived from them. A major research issue, then, is the definition of models of this character.

One can conceive of numerous other preprocessing techniques all designed to reduce the amount of on-line computation required to support high-fidelity testing. Indeed the concept of complete returns calculations prior to testing carries the technique of preprocessing to its limit and we are thus presented with an open-loop system where the sequence of orders is predictable and the responses can be predetermined.

Open-loop operation requires for its success first, that orders to SETS be predictable. If a sensor is asked to scan a portion of the sky according to well-defined parameters then the time history of the sensor returns in the presence of an attack can be precalculated and stored. This mode of testing requires a large mass of storage and sufficient bandwidth to bring in the returns data at the required rate. While the latter requirements involve "merely" additional hardware, the imposition of the requirement of "predictability" on the RTP is too severe. For SETS operating as a development aid, RTP algorithms often change, errors are found in the program, or process timing changes so that orders sequences cannot be assumed to remain constant. This is especially true if the real-time process is distributed over asynchronously operating processes. To be a viable system development aid SETS must maintain its closed-loop character to limit the computer and manpower resources which would otherwise be required.

To summarize, SETS operation in real time is certainly the most desirable mode of testing but the combined requirements of providing high-fidelity responses and closed-loop operation for multiple processes renders the concept useless. Only if one or more of the above constraints can be relaxed, or if the SETS process itself can be broken into distributed processes paralleling the distribution of the RTP is there any hope of carrying out real-time testing of anything more elaborate than the simplest RTP configuration.

Interrupted Real Time -- In interrupted real-time (IRT) operation the RTP operates in spurts of real time. During each of its computation cycles it typically prepares new commands for the sensors under its control and processes the returns from previous commands. If SETS can prepare the returns in the time allotted to it the engagement will proceed uninterrupted. If, however, the SETS process notes that it is falling behind in its calculation it will call for interrupted real time.

SETS knows when its responses are due since it typically simulates defense sensors which respond according to well-defined rules. If IRT is called for, a special signal is sent to the real-time processor to suspend all of its computations and place itself into a wait state until a restart signal is received from SETS, sent when it has caught up with all of its back work. At this time the RTP will pick up where it has left off and communications interchange will restart in a normal fashion.

Implementation of IRT in the BMDATC Baseline System using the ASC/7600 computer hookup required an interface clock readable by both machines and elaborate interface software to clock returns out of the SETS system at the appropriate times and bring in commands generated by the RTP in the ASC. The interface software also noted if the SETS process was nearing a missed deadline and provided the mechanism to enter and exit IRT.

Real time cannot be perfectly interrupted. Because of propagation times and logic requirements it took up to 50 microseconds before the ASC terminated operation following an IRT signal in the baseline system. Furthermore, the transmission of data to or from a peripheral device must be completed before the interrupt can take effect. Rotational storage device positioning and access time imprecision also contribute to the uncertainties of interrupted real-time operation. All of these uncertainties existed in an environment where the computers were colocated; no delays were introduced because of remoteness or long line length.

In a distributed system we are no longer concerned with only the start/stop uncertainties of a single process but also those which develop between the subsystems of the RTP itself. It is clear that a netted system must be completely stopped if one of its components is to be stopped. This follows from the observation that component A must not be permitted to calculate past the point in time where it receives a communication from component B, assumed to be halted by an IRT interrupt. Since interprocess communications occur, generally, at unpredictable intervals, it follows that the only sure way to maintain timing integrity in component A is to halt its operation when B is halted. Therefore the complete net must be halted if SETS lingers in response to one element of the net.

For IRT to operate successfully in such an environment, the individual processors must be prepared to halt instantaneously upon receipt of an interrupt signal. Communications handlers must be prepared to suspend operations following completion of a current transfer of data. The system designers must be prepared to accept as negligible the timing uncertainties that accrue or be able to convincingly argue that they somehow average out. The hardware developers must be prepared to incorporate additional components to respond to start/stop signals and the software developers must be ready to put up with unrepeatable test runs. On the face of it, IRT presents as many problems as real-time testing; however, the interrupted real-time mode of testing appears more within our reach simply because it has been accomplished before, albeit on a much reduced scale than that being proposed with a distributed system.

Repeatable Real Time -- A testing strategy that has never been attempted but appears to be feasible is one which we call "Repeatable Real Time" (RRT). RRT operates exactly as does real-time testing with one exception. The entire returns stream from SETS is recorded onto a high-speed storage device, each set of returns being stored with its associated order. When it is observed that the computing load on SETS will force it to miss a returns deadline, the entire RTP is stopped but SETS is permitted to continue processing its complete input queue of orders. All the while the returns (and orders) are recorded on the temporary file.

When the input queue is empty the complete simulated engagement is restarted from the beginning and now SETS has at its disposal a file of returns for the initial set of RTP orders. SETS is thus relieved of computing the responses in real time for these orders but may merely retrieve the returns from the temporary file and transmit them to the RTP at the appropriate instant in time.

In effect the complete test consists of a series of subtests, each lasting longer than the previous and adding to the file of usable returns. Note that each subtest is in real time but represents only a portion of the complete engagement time simulation. RRT testing is feasible only if certain conditions are true:

1) The orders stream from the RTP to SETS is identical (or nearly identical) from sub-test to sub-test. Because of synchronization difficulties, it may be impossible to generate identical order streams to SETS; in successive runs SETS can be programmed to handle interchanged, mixed or slightly permuted orders since the order is stored on the temporary file along with the returns. New orders which might appear in the stream that hadn't in the past could be handled in a variety of ways. Simple "null-returns" could be prepared requiring minimal SETS resources or the full SETS model could be invoked for a high-fidelity response. The answer to this problem should await the outcome of some preliminary feasibility experiments addressing other aspects of RRT.

2) SETS resources required for accessing returns from the temporary file must be substantially less than required to calculate the response in real time. Of course RRT makes no sense at all if to locate and read returns from a mass storage device takes longer than to recalculate them. But the access time should be substantially less so that for each restart a significant number of new orders can be processed before SETS is forced to stop. Only if this is true can the number of restarts be held to a manageable few.

From the above discussions it should be apparent what the advantages, disadvantages and critical issues of RRT testing are. Clearly, the major goal of real-time, high-fidelity, closed-loop testing can be achieved if repeatable real time is proven feasible. No mechanism to restart a stopped process in the middle of an engagement is required as in interrupted real time and no uncertainties accrue concerning time delays in process stopping, data transfers or storage device positioning.

A potentially serious disadvantage of RRT testing is in the amount of elapsed "wall clock" time required to complete a test. In the limiting case, if SETS is able to process only a single order before stopping and reinitiating the test then the elapsed time to complete a test could be enormous, well beyond the MTBF of the processors involved. However, the most serious disadvantage appears to be the imposition on the RTP of the requirement to generate successive, identical order streams. In a distributed process where one or more components in the test environment may be actual system components while the rest may reside in an emulator and others simulated by SETS the time coordination of the various order streams may not be possible.

A summary of the critical issues with respect to RRT testing identifies a number of potential experiments which might be usefully conducted:

1) How does the time required for SETS to compute high-fidelity returns compare to the time needed to access and locate the same prestored returns? No single answer can be given to this question since the computation portion is highly dependent upon the sensor characteristics, the attack configuration and other environmental effects while, it is hoped, the data retrieval time remains relatively constant.

2) Can a distributed RTP be caused to reinitiate its calculations in a way to cause its communications with the "external world" to be repeatable? The answer to this question will determine the feasibility of repeatable real-time testing.

Non-Real Time -- Many tests are required of an evolving data processing system that have no real-time requirements associated with them. In this category we place algorithm evaluation, interprocess communication protocol, determining the validity of functional models, and verifying the correctness of control paths. Most of these test configurations arise at early levels of system development. They are not, however, to be considered unimportant since at these stages many errors can be located with less costly consequences than if detected later.

The critical issues associated with non real-time testing are in the realm of test configuration and test management. These issues are properly part of those considered under the category of "V and V" and are discussed in another section.

A summary of the major issues associated with each type of SETS operating environment is shown in Table 2.

4.2.2 SETS Configurations

The problem of matching an applications requirement to a computer structure is a difficult task. The major thrust of distributed data-processing research is directed toward devising a method describing the mapping process for a variety of applications. Configuring the SETS applications requirement to an appropriate structure is made more difficult still since its configuration must certainly depend upon the ultimate distribution of the real-time process itself. Therefore a degree of freedom is taken from the SETS designer in that his process architecture must somehow follow or at least not be incompatible with the real-time process structure.

Suppose that the real-time process has been carefully subdivided into subprocesses and then been allocated to nodes of the defense system network. The activity of building the software then begins and the need for SETS to assist in testing arises. If a well-disciplined methodology is used to develop the RTP there will undoubtedly be system-level simulations, functional representations of modules evolving into algorithmic code, simulations by which message interactions and control flow can be examined; in short, a wide spectrum of coded products which need to be tested using SETS.

TABLE 2. DISTRIBUTED SETS OPERATING ENVIRONMENT AND CRITICAL ISSUES

Sets Operating Environment									Issues									
Timing				Fidelity		Operation												
RT	IRT	RRT	Non-RT	High	Low	Closed Loop	Open Loop	Why This Type Testing?	Sets Computer Power	Good Low-Fidelity Models	Data Rates RTP-SETS	RTP Repeatability	RTP Stop, Synchronize	Long Preprocessing	Large Mass Store	Long Test Times	Data Rates Mass Store	Traceability
X				X		X		Most desirable test configuration	X		X							
X				X			X	To maintain real time with minimum computing power				X		X	X		X	
X					X	X		To maintain real time with minimum computing power		X	X							
X					X		X	Only if real time unobtainable in any other way				X			X		X	
	X			X		X		To maintain hi-fi testing in closed loop					X			X		
	X			X			X	To reduce test time if closed loop calculations lengthy				X	X	X		X		
	X				X	X		To test *logic flow if cannot operate in real time*		X			X			X		
	X				X		X	Probably never be done				X	X			X		
		X		X		X		T^ carry out high-fidelity real-time testing								X		
		X		X			X	Not applicable				X						
		X			X	X		To reduce testing time				X				X		
		X			X		X	Not applicable				X						
			X	X		X		When timing is unimportant, but fidelity is (sub-system testing)								X		X
			X	X			X	Only if on-line processing is expensive; multiple tests				X		X	X			X
			X		X	X		Early development phases of individual modules		X								X
			X		X		X	Only if on-line processing is expensive		X		X		X	X			X

We can envisage a distributed process in which one function is coded and is being tested in its analytic form while interfaced to other modules represented functionally. Typically, the development process will be uneven in that the same level of detail is impossible to maintain over all processes. So, at a given instant in time we can view the partially developed system as containing modules of varying fidelity, communicating with each other and each possibly requiring interaction with SETS, each at its own fidelity level.

What does all this mean to the SETS designer? The implications are staggering. Let's try to get a handle on the problem by simplifying it a bit.

Centralized SETS -- The easiest simplication is to assume that SETS will operate in a single computer in a physical environment co-located with only a few of the distributed sub-processes. Figure 6 illustrates such a configuration where each processor communicates with each of the others as well as with the SETS program. In this simple configuration the SETS designer must address a number of critical problems, namely:

- Managing multipl϶ input and output ports
- Establishing policy for preemption of SETS sub-processes
- Synchronizing timing if interrupted real-time operation becomes necessary
- Scheduling of asynchronous messages from the world external to the RTP

In a realistic software development environment no constraints should be placed on the level of model fidelity resident at any one time in a real-time process. Therefore, superimposed on the above design requirements is the additional requirement to provide a collection of SETS modules representing a wide spectrum of fidelity. Implied in this requirement is that SETS be a simulation system in that it be able to model all system phenomena at any level of fidelity; that support tools be available to simplify the activity of configuring the appropriate SETS construct.

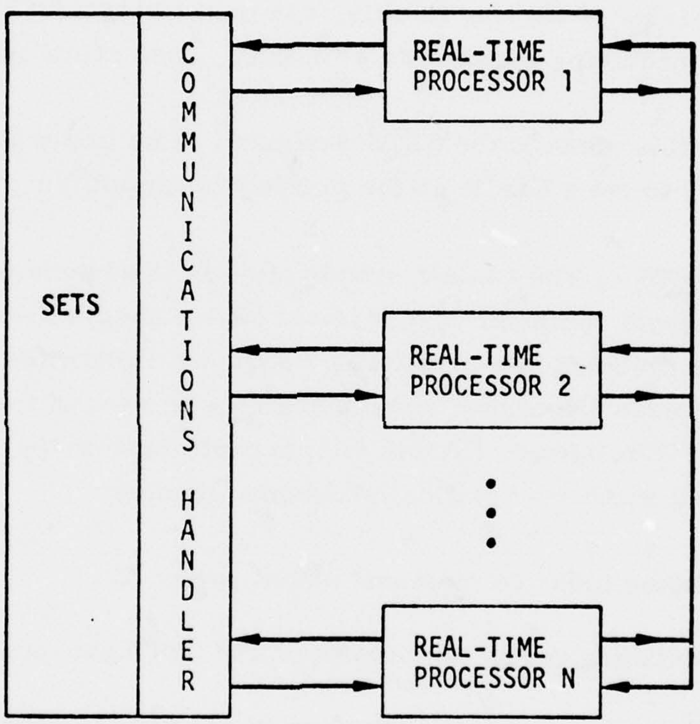

Figure 6. Centralized SETS Configuration

Design solutions to the centralized SETS approach must describe how the required data rates will be maintained to properly simulate the world external to the RTP. It should be obvious that the more elaborate the RTP becomes, that is, the more sub-processes in the configuration, the more difficult it will be to achieve the required communications rates.

The question of synchronizing timing has been discussed earlier and various alternatives presented along with their implementation issues. A centralized SETS configuration does nothing to alleviate the synchronization problems: they remain a major critical design issue.

The critical issues requiring resolution before a centralized SETS system can be successfully realized can be categorized briefly as 1) partitioning and scheduling of the SETS processes; 2) supporting the response requirements of high input traffic; 3) interfacing processes of varying levels of fidelity; 4) coordinating timing between the RTP and SETS.

Distributing SETS -- An alternative SETS design to that of a centralized configuration would have the SETS processes themselves distributed over several processors. Just as there is wide freedom of distributing the real-time processes with respect to the design payoffs of growth and survivability, there are at least as many ways of distributing SETS. Many of the payoffs which guide the RTP design are applicable to SETS but some are not. A deployed RTP needs to be highly reliable. The SETS system, used only in a test environment, does not need to supply backup modules or redundant components for reliability purposes; if a SETS component fails the test can be reinitiated. There are some requirements on SETS, however, which are more stringent than corresponding RTP requirements, to be discussed later.

It is felt that the decomposition and allocation of the SETS processes should, in some sense, follow and parallel that of the RTP. In fact, there could be a SETS processor identified for each real-time processor and the one-for-one correspondence would extend, of course, to the models within the processors. Figure 7 illustrates such a testing configuration. In addition to the one-for-one processor assignment, provision must be made to control time synchronization during testing. This problem has been discussed in earlier sections and will not be repeated here. However, the need for a mechanism to provide start/stop signals in certain test environments implies a central source supervising those functions.

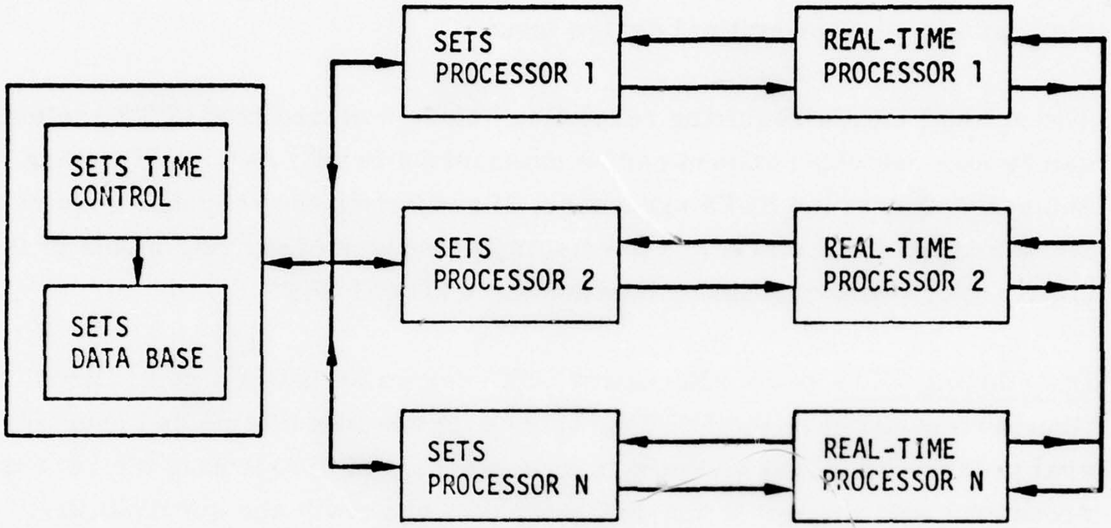

Figure 7. Distributed SETS Configuration

There is also the problem of the SETS data base. The data quantities used by SETS in modeling the real world change for two reasons. As the engagement proceeds, the changing attack configuration must be reflected in the quantities in the data base at any instance. Furthermore, actions taken by the RTP can affect the battle environment and this information must be incorporated into the SETS data base. Since there is now a need for access by different processes into, conceptually at least, a common data base, there now arise the myriad problems of managing distributed data bases. Since the SETS data base must provide a true picture of a rapidly changing world, a function is therefore required to update and maintain the data in accordance with the advance of time. This function is not associated with any real-time process; it is wholly a stand-alone SETS program function. Since it is quite likely that the SETS data base will itself be physically distributed, the maintenance of data time integrity over geographically spread bases must take into account transmission delays, multi-access conflicts, and lockouts. This brings us to a major problem associated with distributing SETS and its data over a wide geographical area.

Message delays between elements of the RTP must be considered to be part of the process itself. That is, during a war, message delays and loss of data will most certainly occur because of the physical limitations of distributed processes. The RTP design demands that logic within the real-time process exist to recognize these occurrences, and respond in such a way as to minimize their effects. It is, however, a basic axiom of "SETSmanship", of testing using an environmental simulation, that the external world be presented in as realistic a manner as possible. Message delays incurred in the SETS process cloud the realism that SETS is attempting to present and could cause events to occur which would not arise when the RTP operates in a war environment as opposed to a test environment. This is a case where a design requirement on SETS is more stringent than on the RTP, namely the minimization of message traffic delay.

A one-for-one SETS distribution must account for dynamic RTP configuration in the face of overload or other degrading pressures. If processes can be reassigned to processors in response to failure, overload, or for other beneficial reasons, then the corresponding SETS process must also be reconfigured.

To summarize, the critical issues associated with a simple policy of one-on-one assignment of a SETS processor to an RTP processor can be categorized as 1) managing a distributed SETS data base; 2) maintaining time synchronization in testing; 3) removing SETS message delays and data access conflicts; 4) permitting flexibility in reconfiguring the real-time process.

A Distributed SETS Design Discipline -- A list of the critical design issues of SETS in a distributed RTP environment brings to light many serious testing problems. However, there is a concept that has the potential of alleviating many apparent SETS design problems. Simply stated, that concept is:

> Account for testing needs and testing interfaces at all levels of RTP design.

This concept is not alien to any theoretical aspect of a top-down design discipline of distributed systems. To the contrary, "testability" must be one of the major unstructured, nonfunctional requirements guiding the design of BMD systems. That abstract concept must be incorporated at each design level, and must be considered in each decision so that design validity can be established at every step.

Past experience has shown that, in practice, design testing is often undertaken as a separate activity by a validation staff on a test article conceived with anything but testability in mind. No special hardware ports are supplied, no operating system functions are provided, no tradeoffs are made in the interest of easing the test process. In BMD systems, the justification for this type of design goes something like this: our system (the real-time process) cannot afford to accommodate any testing interfaces because the overhead sustained reduces our real-time capability; testing interfaces are not needed in a war-ready configuration.

It is true that designing in "testability" is not simple, and that some system resources must be given over to its achievement. We believe that the overhead can be kept to a minimum if testing problems are kept in mind from the beginning and contend that by building testability in from the onset the following benefits will accrue:

- Greater visibility into the evolving real-time process
- Increased capability for substantive testing
- Reduced total system life cycle costs
- Clearer resolution of testing problems such as configuration of SETS processes.

If we agree that testability is a desirable attribute to incorporate into all levels of real-time process design, how are the SETS critical design problems thereby reduced or alleviated? To a large degree they are alleviated by forcing them to be considered as the distributed process is being designed. In our ideal model, each real-time process module is designed not only from functional requirements, timing constraints, analytic algorithms or whatever else is required at the level of interest, but also from the environmental model, algorithms and timing information needed for its testing. The test driver, SETS, must be considered as part of the RTP. As total RTP requirements become more detailed the hardware selection decisions will then include, by default, hardware required for testing.

By the same process handling of SETS data bases, inter-SETS communications needs and other pressing SETS-related problems will be resolved during RTP design.

By no means have we resolved all SETS design problems when we put them into someone else's realm of concern. But what we have done is forced them to be considered early during design and guaranteed that no RTP system will evolve that cannot be validated. The ramifications of this concept leads to a discipline in which the SETS configuration evolves as does the real-time process configuration, under the stimulus of the global requirement, testability.

As must other global, or unstructured, requirements, "testability" must be partitioned into more precisely defined concepts and then a set of means identified to satisfy those concepts. These remain as major issues to be resolved if this design discipline is to be used as it should.

REFERENCES

1. Freeman, P., "The Context of Design," Tutorial on Software Design Techniques, P. Freeman and A. I. Wasserman, eds., IEEE Catalog No. 76H1145-2 C, IEEE Computer Society, Long Beach, CA, 1976, 2-11.

2. "Software Requirements Engineering Methodology," TRW Report 27332-6921-024, (CDRL C011 of DASG60-75-C-0022), 1 September 1976, Uncl.

3. "Process Design Methodology Description," TI Report H750202D-A, (CDRL C00A of DAHC60-72-C-0156), Vol. V, December 1975. draft, Uncl.

4. Requirements Engineering and Validation System User's Manual, TRW Report 27332-6921-022, 15 July 1976 (draft), Unclassified.

BIBLIOGRAPHY

1. Agajanian, A. H., "A Bibliography on System Performance Evaluation," Computer, Vol. 8, No. 11, November 1975, pp. 63-74.

2. Anderson, G. A. and Jensen E. D., "Computer Interconnection Structures: Taxonomy, Characteristics, and Examples," ACM Computing Surveys, 7, 4 (December 1975), 197-214.

3. Balkovich, E. E., et al., Research Towards a Technology to Support the Specification of Data Processing System Performance Requirements, General Research Corporation CR-5-685, July 1976.

4. Buley, E. R., et al., Adaptive V&V Final Report, General Research Corporation CR-5-708, January 1977.

5. Copper, D. W. and Stone, R. L., Hierarchical V&V Software Functional Design, Final Report, Vol. 2, Part 1, "Hierarchical SETS (H-SETS)," General Research Corporation CR-7-626, March 1976.

6. Copper, E. W., Adaptive Learning Requirements and Critical Issues, General Research Corporation CR-4-708, January 1977.

7. Franta, W. R. and Maly, K., "A Framework for Investigating a Class of Distributed Processing Algorithms," IEEE Trans, on Systems, Man and Cybernetics, Vol. SMC-7, No. 8, August 1977, pp. 597-604.

8. Morgan, D. E., et al., "A Computer Network Monitoring System," IEEE Trans. on Software Engineering, Vol. SE-1, No. 3, September 1975, pp. 299-311.

9. Palmer, D. F., et al., Hierarchical V&V Software Functional Design, Vol. 1, "Hierarchical V&V Methodology Description, Final Report, General Research Corporation CR-7-626, March 1976.

10. Ripley, C. D., "Program Perspective: A Relational Representation of Measurement Data," IEEE Trans. on Software Engineering, Vol. SE-3, No. 4, July 1977, pp. 296-300.

11. Saib, S. H., et al., Advanced Software Quality Assurance Final Report, General Research Corporation CR-6-720, March 1977.

12. Sholl, H. A., Booth, T. L., "Software Performance Modeling Using Computation Structures," IEEE Trans. on Software Engineering, Vol. SE-1, No. 4, December 1975, pp. 414-420.

13. Svobodova, L., "Computer System Measurability," Computer, Vol. 9, No. 6, June 1976, pp. 9-17.

APPENDIX A

OUTLINE OF CURRENT BMDATC SOFTWARE DESIGN SYSTEM (SDS)

The following outline of SDS design phases was derived mainly from references 2 and 3, as interpreted by GRC personnel. We believe that the outline is in full agreement with the documentation, although some misinterpretations might exist.

SDS has two major design activities: Software Requirements Engineering (SRE) and Process Design Engineering (PDE). The objective of the SRE methodology (SREM) is to provide an explicit sequence of steps, with phases for management review and analysis, which lead to a formal specification of requirements for software to meet a specific BMD system specification. The resulting formal specification, termed a Process Performance Requirements (PPR) Specification, is intended to be an unambiguous and complete statement of requirements in an implementation-free (nonconstraining) context and format.

The inputs to the SRE are assumed to be a system specification and an interface specification. The formats and contents are not explicitly defined, this definition was an objective of the Data Processing Subsystem Engineering (DPSE) program, which is not active at present. The input specifications are not assumed to be complete (nor necessarily correct) at the beginning of SRE; they are to evolve in detail and quality during SRE, benefitted both by SRE and other subsystem design activities.

The PDE Methodology (PDM) is intended to be a complete methodology for the design, implementation, and test of a software process to meet a PPR. PDM assumes that the DP hardware configuration is preselected, and that the target DP hardware, compiler, loader, etc., and interfaces are available in time to support final design phases.

Both SREM and PDM rely heavily on the support of a special language and special support software (for cataloging, analyzing, and displaying the evolving design). SREM has the Requirements Statement Language (RSL) and is supported by the Requirements Engineering and Validation System (REVS), in particular by the REVS Requirements Analysis and Data Extraction (RADX) procedures.[4] PDM has the Process Design Language (PDL) and is supported by the Process Design System.*

Both SREM and PDM freely admit to phases which require "creativity" and iteration. The SDS design approach, therefore, provides more a sequence of things to do than descriptions of how to do the things. The outline given below consequently contains essentially all of the available information about the approach except for details of how to use various tools.

1. SRE Phase 1: Definition of DPS Elements (Define processing flow connectivity (using "requirements nets," R-nets) and data hierarchies)

1.1 Identify DPS message communication interfaces with other subsystems.

1.2 Define unique names for interface messages and message contents hierarchies (to develop in detail as design proceeds).

1.3 For each input (to DPS) interface establish at least one R-net, (usually) at least one processing step (alpha) for communications, and at least one unique alpha for each message type. (The latter alphas are joined by an exclusive OR node, assuming each message type requires unique processing).

*Vols. I-IV of Ref. 3.

1.4 Further evolve the R-nets through analysis of the DPS specification and application of engineering creativity. (Refine/create R-nets).

1.4.1 Construct processing threads to perform described functions.

1.4.2 Construct processing sequences at DPS output interfaces (tracing backwards, if necessary) to provide specified output message types.

1.4.3 Identify "entities"--things or groups of things in the external world about which the DPS must collect, process, and maintain data.

1.4.4 Define unique names for entities and contents of their data hierarchies (to develop in detail as design proceeds).

1.4.5 Define processing steps necessary to create and destroy instances of entities.

1.4.6 Construct processing sequences and data structures corresponding to specification sentences (e.g., relate data entities to nouns and alphas to verbs).

1.4.7 Define independent data files (files not contained in established hierarchies) as necessary.

1.4.8 Analyze and restructure R-nets as necessary to correctly define potential asynchronous processing wherever possible, using engineering experience.

2. SRE Phase 2: Evaluate the Phase 1 Requirements Kernel

2.1 Enter the requirements into REVS, if not already entered.

2.2 Invoke the REVS Requirements Analysis and Data Extraction (RADX) procedures to check the R-net structures and enabling conditions and data hierarchy structures.

3. SRE Phase 3: Completion of the Functional Definition

3.1 Define data transactions of each alpha. (Add alphas or redefine alphas into subnets as necessary).

3.1.1 Abscribe known data elements to inputs and outputs of alphas. (Generally, data which is global to multiple R-nets or is locally used in interface messages is known).

3.1.2 Using RADX, evaluate data transactions to find alphas with no input or output data and inconsistent create/destroy activities.

3.1.3 Further define data relationships to be consistent with R-net structure and alpha definitions.

3.2 Define and check all required data attributes (local/global, type, use by models of alphas (termed betas and gammas), and range of enumerator variables) as possible without constructing beta/gamma simulation models.

4. SRE Phase 4: Development of Functional Models

4.1 Develop functional models (betas) of alphas to implement data flow required for an executable functional simulation. (This requires engineering creativity).

4.2 Apply RADX to analyze new data definitions required by the betas.

4.3 (Assumed) Exercise the functional model to investigate dynamic consistency of logic and further define data requirements.

5. SRE: Develop management segment of Requirements. Provide descriptive material to be used for traceability and information purposes during further development.

6. SRE: Develop Analytic Models

6.1 Develop analytic models (gammas) of alphas to provide realistic data transformations in nonreal time.

6.2 (Assumed) Exercise the analytic model to show the feasibility (existence of a design which can meet specified requirements).

7. SRE: Identify Performance Requirements

7.1 Identify all performance requirements (accuracy, port-to-port timing, global effect) through analysis of the system specification.

7.2 Name and provide traceability references for all performance requirements identified.

8. SRE: Locate Test Points

8.1 For each performance requirement, locate a test point on the appropriate R-nets or subnets. (Test points become "validation points" which terminate "validation paths" to be defined by steps below).

9. SRE: Define Initial Validation Paths

9.1 Formulate an initial version of a test to be performed at a test point for each performance requirement.

9.2 Determine data required for the "initial test formulation and compare with data available at the test point."

9.3 Define data required for each test and not available at the test point and locate points on R-nets and subnets to collect this additional data; these points are labeled as "validation points."

9.4 Define an initial "validation path" to describe the structure of each set of validation points (including the terminating test point) defined above for each initial test formulation.

10. SRE: Complete Validation Path Definition

10.1 Finalize each test formulation and refine the initial declaration of validation paths.

10.2 Encode each test as a PASCAL procedure and enter into the PPR data base.

11. PDE Phase 1: Requirements Analysis

11.1 Establish a traceable mapping of the PPR Requirements Nets into data processing tasks requirements structured to effectively utilize the architectural characteristics of the real-time data processor. In particular, assuming a defined hardware configuration, restructure the R-nets to group processing steps (alphas) into tasks on the basis of commonality of:

- Suitable processor types and capabilities (where the hardware configuration is assumed to contain more than one type of processor)
- Scheduling frequency and conditions
- Memory organization (assuming a memory capability hierarchy to exist)
- Potential parallelism of different steps acting on one data entity instance of concurrency of identical steps acting on many instances of a data entity assuming the hardware configuration to contain more than one processor architecture)
- Real-time operating system requirements: tasking and data hierarchy, scheduling fidelity, and memory management.

This activity may produce more than one candidate process configuration.

11.2 Establish a performance/quality test structure for use throughout the remaining process design phases.

11.2.1 Design first-level data processor observer process (DPOP).

11.2.2 Establish initial hierarchy of functional to analytical RTSW/H-SETS interface requirements and scenario requirements.

11.3 Provide cost model of projected PD phases and initial estimates of time, funding constraints.

12. PDE Phase 2: Top Level Design

12.1 Assign tasks identified during phase 11.1 to computer processors, establishing the communication mechanism among these tasks, and defining the scheduling rules for providing real-time control and synchronization for the system.

12.2 Develop and test a functional model of this top-level structure.

12.2.1 Directly from PPR, for each task:

- Identify minimum data structure for validation point data collection
- Delineate PPR scheduling rules (linked to input stimuli rate) for executing tasks.

12.2.2 Estimate resources required (memory and execution time--for target hardware):

- Initially guess, later update.

12.2.3 Define data files for major communication paths between tasks.

12.2.4 Code the functional mode in PDL to be executed initially via the PDS simulation executive.

- This defines candidate top-level real-time process design.

12.3 Test the model against performance requirement using DPOP from phase 11.2.

12.4 Iterate until a particular requirements mapping satisfies all performance requirements.

13. PDE Phase 3: Sequencing Logic Design (Decomposition Design)

13.1 Viewing each task as an isolated program in the system to be placed into execution on a particular processor, decompose each task into subtasks on the basis of commonality of:

- Input/output
- Timing and scheduling
- Logic characteristics

13.2 Prescribe the sequencing (or decision) logic for selective execution of the subtasks of each task.

13.3 Model the subtasks with stubs, then functional models using truth data from SETS.

13.4 Decompose the data file structure to best match specific design configurations.

13.5 Extensively test this "hybrid" system:

- Interfacing control functions are correct
- Sequencing logic of each task executes the models in desired order.

13.6 Reiterate 13.1 through 13.5 at lower and lower levels (e.g., first treat subtasks as tasks) until the "bottom" subtasks correspond to computational algorithms (in most cases).

14. PDE Phase 4: Detailed Design

14.1 Replace subtask models form phase 3 with actual algorithm code.

14.1.1 Structure algorithms to meet logic requirements and optimize use of computer.

14.1.2 Test and iterate algorithm designs within higher-level model structures to provide performance consistent with higher-level design requirements.

15. PDE Phase 5: Process Design Validation (concurrent with phase 14)

15.1 Extensively test the evolving process against scenarios specified by the PPR.

15.2 Evaluate performance of the process against criteria established in the PPR and embedded in the observer process.